Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA481 Programmering av inbyggda system D
EDA486 Programmering av inbyggda system Z
DAT016 Programmering av inbyggda system IT
DIT152 Programmering av inbyggda system GU
LEU500 Maskinorienterad programmering, DAI,EI,MEI

Måndag 13 april 2015, kl. 14.00 - 18.00 Johanneberg/Lindholmen

---

Examinator
Roger Johansson

Kontaktpersoner under tentamen
Johanneberg: Roger Johansson, tel. 772 57 29
Lindholmen:Lars Bengtsson, tel 772 84 41

Tillåtna hjälpmedel

Häftet

Instruktionslista för CPU12

Inget annat än rättelser och understrykningar får vara införda i häftet.
Du får också använda bladet:

C Reference Card

samt *en* av böckerna:

Vägen till C,
Bilting, Skansholm

The C Programming Language,
Kerninghan, Ritchie

Endast rättelser och understrykningar får vara införda i boken.

Tabellverk eller miniräknare får ej användas.

Lösningar
anslås senast dagen efter tentamen via kursens hemsida.

Granskning
Tid och plats anges på kursens hemsida.

Allmänt
Siffror inom parentes anger full poäng på uppgiften. **För full poäng krävs att**:

- redovisningen av svar och lösningar är läslig och tydlig. Ett lösningsblad får endast innehålla redovisningsdelar som hör ihop med en uppgift.
- lösningen ej är onödigt komplicerad.
- du har motiverat dina val och ställningstaganden
- assemblerprogram är utformade enligt de råd och anvisningar som givits under kursen och tillräckligt dokumenterade.
- C-program är utformade enligt de råd och anvisningar som getts under kursen. I programtexterna skall raderna dras in så att man tydligt ser programmens struktur.

Betygsättning
För godkänt slutbetyg på kursen fordras att både tentamen och laborationer är godkända.

Tentamenspoäng ger slutbetyg enligt:
(EDA/DAT/LEU):
20p≤betyg 3 < 30p ≤ betyg 4 < 40p ≤ betyg 5
respektive (DIT):
20p≤betyg G < 35p ≤ VG

Uppgift 1 (14p) *Kodningskonventioner (C/assemblerspråk)*

I denna uppgift ska du förutsätta samma konventioner som i XCC12, (bilaga 2).

Vi har en C-funktion enligt följande:

```
void  fill( const unsigned int size, char pattern, const char *start )
{
   unsigned int count = size;

   while( count-- )
   {
      *start++ = pattern;
   }
}
```

a) Översätt funktionen `fill` till motsvarande subrutin `_fill` i HCS12 assembler, subrutinen ska kunna anropas från en C-funktion. Speciellt ska du börja med att beskriva aktiveringsposten, dvs. stackens utseende i funktionen. Visa tydligt riktningen för minskande adresser hos aktiveringsposten (6p).

b) Följande C-deklarationer har gjorts på ”toppnivå” (global synlighet).

```
char     this_pattern = 0x33;
char     this_memory[100];
```

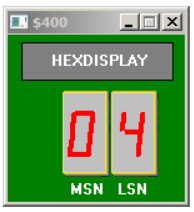
Visa motsvarigheten i assemblerspråk för HCS12 (4p).

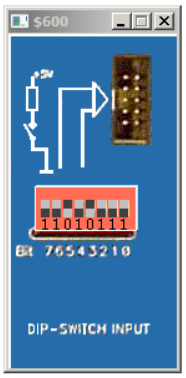
c) Med förutsättningar enligt uppgift a) och b) ovan, visa hur följande funktionsanrop kan översättas till assemblerspråk för HCS12 (4p)

```
fill(18, this_pattern, &this_memory[0] );
```

Uppgift 2 (8p) *In och utmatning beskriven i C*

I denna uppgift ska du bland annat demonstrera hur absolutadressering utförs i C. Visa speciellt hur preprocessordirektiv och typdeklarationer används för att skapa begriplig programkod.





En 8-bitars strömbrytare är ansluten till adress 0x600 och en displayenhet som visar en byte i form av två hexadecimala siffror är ansluten till adress 0x400 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void FindFirstZero( void )
```

som läser av strömbrytaren och indikerar den minst signifikanta nollställda biten genom att skriva dess position, räknat från höger, till displayenheten. Om exempelvis bitarna b_3 och b_5 utgör nollställda strömbrytare ska positionen för b_3, (dvs. 4) skrivas till displayenheten.

Om ingen strömbrytare är nollställd ska siffran 0 skrivas till displayen.

Uppgift 3 (8p) *Programmering med pekare*

Standardfunktionen `strspn` kan beskrivas på följande sätt:

```
int strspn ( const char * str1, const char * str2 );
```

Returns the length of the initial portion of *str1* which consists only of characters that are part of *str2*.

Parameters
str1 C string to be scanned.
str2 C string containing the characters to match.
Return value
The length of the initial portion of *str1* containing only characters that appear in *str2*. Therefore, if all of the characters in *str1* are in *str2*, the function returns the length of the entire *str1* string, and if the first character in *str1* is not in *str2*, the function returns zero.

Exempel på användning:

```
int main()
{
  char string[]="7803 Elm St.7";
  printf("The number length is %d.\n",strspn(string,"1234567890"));
  return 0;
}
```

Utskriften ska då bli: ”The number length is 4”.

Din uppgift är att, i C, skriva en egen definition av funktionen `strspn`. Du får inte använda dig av indexering, utan måste utnyttja pekare. Du får inte anropa någon annan standardfunktion. Du måste alltså skriva all kod själv.

Uppgift 4 (6p) *Assemblerprogrammering*

Följande funktion `bitMask` finns given som en specification i “C”. Funktionen bestämmer en så kallad “bitmask” baserad på inparametrarnas värden.

```
unsigned char  bitMask( unsigned char bit_number, unsigned char bit_value )
{
   static   unsigned char  set[]   = {0x80,0x40,0x20,0x10,8,4,2,1};
   static   unsigned char  clear[] = {0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE};

   if( ( bit_number > 7 ) || ( bit_value > 1 ) )
      return 0;
   if( bit_value )
      return ( set[bit_number] );
   else
      return ( clear[bit_number] );
}
```

Skriv motsvarande funktion BITMASK i assemblerspråk för HC12. Observera att i denna uppgift ska du **inte** ta hänsyn till kompilatorkonventioner för XCC12. I stället skickas parametrarna enligt:

`bit_number` i register B
`bit_value` i register A

returvärde från funktionen i register B.

Uppgift 5 (14p) *Maskinnära programmering i C*

a) (6p)

I den sista laborationen konstruerade du en funktion `hold` som användes för att fördröja exekveringen så många millisekunder som parametern angav. Funktionen använde sig av en statisk variabel `tick` av den definierade heltalstypen `time_type`. Denna variabel räknades upp av realtidsklockan vid varje klockavbrott. Tiden mellan varje klockavbrott var 10 ms. Funktionen `hold` låg i en modul som består av filerna `clock.h` och `clock.c`.

Din första uppgift är nu att konstruera en ny funktion `hold_timeout` som är snarlik funktionen `hold`. Du får anta att den nya funktionen placeras i samma modul som `hold` så att den har tillgång till variabeln `tick` och typen `time_type`. Funktionen `hold_timeout` skall ha två parametrar. Den första skall vara en pekare till en funktion som är parameterlös och som returnerar ett sanningsvärde (0 eller 1). Detta sanningsvärde anger om en viss händelse inträffat (1) eller inte (0). Funktionen `hold_timeout` skall normalt vänta tills funktionen (vars pekare skickats som parameter) returnerar ett värde som är sant. Den andra parametern till `hold_timeout` skall vara av typen `time_type` och ange ett time-out intervall. `hold_timeout` skall vänta högst det antal millisekunder som denna parameter anger. Den andra parametern får vara negativ. I så fall skall det tolkas som att det inte finns någon time-out. Då finns det ingen gräns för hur länge funktionen `hold_timeout` kan vänta. Funktionen `hold_timeout` skall som resultat ge ett sanningsvärde (1 eller 0) som är sant om den återvände för att den händelse den väntat på inträffat och falskt om den återvände för att time-out intervallet löpt ut.

b) (8p)

I ett hus skall ett inbrottsalarm installeras. En infravärmesensor som ger ett avbrott till datorn då en värmekälla detekteras skall användas. Sensorn har ett kombinerat 8-bitars status- och kontrollregister på adressen A024 (hex). Sensorn startas och stängs av genom att bit nr 7 i registret sätts till 1 resp. 0. När en värmekälla upptäcks sätter sensorn bit nr 6 i registret till 1. Om man har satt bit nr 1 i registret till 1 genererar sensorn även en avbrottssignal. Sensorn kan återställas efter ett alarm genom att man sätter bit 6 till 0. Avbrottsvektorn för sensorn ligger på adressen FE20 och minnet här är både läs- och skrivbart.

Den andra uppgiften är att i C skriva ett program som hanterar sensorn och som ger ett larm om sensorn indikerar en värmekälla. För att undvika falskt alarm skall programmet utformas så att det krävs två separata indikationer inom 10 sekunder från sensorn innan larm ges. Du får anta att det finns en färdigskriven funktion `raise_alarm` som man kan anropa för att ge larm.

Du får använda dig av alla de funktioner och makrodefinitioner du skrev i den sista laborationen, t.ex. funktionen `init_clock` som initierar realtidsklockan och ser till att man kan hantera klockavbrott. Din uppgift är alltså att skriva `main`-funktionen samt de funktioner som initierar sensorn och hanterar avbrott från den. Använd dig av funktionen `hold_timeout` från deluppgift a. (Det får du göra även om du inte löst den deluppgiften.) För enkelhets skull får du anta att det finns en färdig avbrottsrutin med namnet `alarmtrap` skriven i assembler. Det enda denna funktion gör är att anropa en funktion med namnet `alarm_inter`. Du måste alltså själv skriva funktionen `alarm_inter`. Du kan däremot inte förutsätta att adressen till avbrottsrutinen är inlagd i avbrottsvektorn. Det måste du göra själv.

Bilaga 1: Kompilatorkonvention XCC12:

- Parametrar överförs till en funktion via stacken och den anropande funktionen återställer stacken efter funktionsanropet.
- Då parametrarna placeras på stacken bearbetas parameterlistan från höger till vänster.
- Lokala variabler översätts i den ordning de påträffas i källtexten.
- *Prolog* kallas den kod som reserverar utrymme för lokala variabler.
- *Epilog* kallas den kod som återställer (återlämnar) utrymme för lokala variabler.
- Den del av stacken som används för parametrar och lokala variabler kallas *aktiveringspost*.
- Beroende på datatyp används för returparameter HC12:s register enligt följande tabell:

Storlek	Benämning	C-typ	Register
8 bitar	byte	char	B
16 bitar	word	short int *och* pekartyp	D
32 bitar	long float	long int float	Y/D

Bilaga 2 - Assemblerdirektiv för MC68HC12.

Assemblerspråket använder sig av mnemoniska beteckningar som tillverkaren Freescale specificerat för maskininstruktioner och instruktioner till assemblatorn, s.k. pseudoinstruktioner eller assemblerdirektiv. Pseudoinstruktionerna framgår av följande tabell:

Direktiv	Förklaring
`ORG N`	Placerar den efterföljande koden med början på adress N (ORG för ORiGin = ursprung)
`L: RMB N`	Avsätter N bytes i följd i minnet (utan att ge dem värden), så att programmet kan använda dem. Följden placeras med början på adress L. (RMB för Reserve Memory Bytes)
`L: EQU N`	Ger label L konstantvärdet N (EQU för EQUates = beräknas till)
`L: FCB N1,N2..`	Avsätter i följd i minnet en byte för varje argument. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FCB för Form Constant Byte)
`L: FDB N1,N2..`	Avsätter i följd i minnet ett bytepar (två bytes) för varje argument. Respektive bytepar ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FDB för Form Double Byte)
`L: FCS "ABC"`	Avsätter en följd av bytes i minnet, en för varje tecken i teckensträngen "ABC". Respektive byte ges ASCII-värdet för A, B, C etc. Följden placeras med början på adress L. (FCS för Form Caracter String)

Lösningsförslag

Uppgift 1a:

_

minnesanvändning	*adress*
start	7,SP
pattern	6,SP
size	4,SP
PC returadress	2,SP
count	0,SP

minskande adress ↓

```
_fill:
   LEAS   -2,SP
;     3 |    unsigned int count = size;
   LDD    4,SP
   STD    0,SP
;     4 |
;     5 | while( count-- )
_fill_2:
   LDX    0,SP
   TFR    X,D
   DEX
   STX    0,SP
   CPD    #0
   BEQ    _fill_3
;     6 | {
;     7 |    *start++ = pattern;
   LDX    7,SP
   LDAB   6,SP
   STAB   1,X+
   STX    7,SP
;     8 | }
   BRA    _fill_2
_fill_3:
;     9 | }
   LEAS   2,SP
   RTS
```

Uppgift 1b:

```
_this_pattern:   FCB    0x33
_this_memory:    RMB    100
```

Uppgift 1c:

```
LDD   #_this_memory
PSHD
LDAB  this_pattern
PSHB
LDD   #18
PSHD
JSR   _fill
LEAS  5,SP
```

Uppgift 2:

```
typedef volatile unsigned char *port8ptr;

#define ML4OUT_ADR  0x400
#define ML4IN_ADR   0x600

#define ML4OUT   *((port8ptr) ML4OUT_ADR)
#define ML4IN    *((port8ptr) ML4IN_ADR)
```

```
void   FindFirstZero( void )
{
   unsigned char pattern, bitpos;
   pattern = ML4IN;

   if( pattern==0xFF  )
      bitpos = 0;
   else{
      for( bitpos = 1; bitpos <= 8; bitpos++ )
      {
         if( (pattern & 1 )==0)
            break;
         pattern >>= 1;
      }
   }
   ML4OUT = bitpos;
}
```

Uppgift 3:

```
int strspn(const char* cs, const char* ct)
{
        int n;
        const char* p;
        for(n=0; *cs; cs++, n++)
        {
                for(p=ct; *p && *p != *cs; p++)
                        ;
                if (!*p)
                        break;
        }
        return n;
}
```

Uppgift 4:

```
;  Data
set:      FCB    $80,$40,$20,$10,8,4,2,1
clear:    FCB    $7F,$BF,$DF,$EF,$F7,$FB,$FD,$FE

MASKBIT:
          CMPB   #7              ; if( bit_number > 7 )
          BHI    maskbit_exit_0
          CMPA   #1              ; if( bit_value > 1 )
          BHI    maskbit_exit_0
          BNE    maskbit_1       ; if( bit_value)
          LDX    #set            ; return ( set[bit_number] );
          BRA    maskbit_2
maskbit_1:
          LDX    #clear          ; return ( clear[bit_number] );
maskbit_2:
          LDAB   B,X
          RTS
maskbit_exit_0:
          CLRB                   ; return 0
maskbit_exit:

          RTS
```

Uppgift 5a:

```
// Tillägg i filen clock.h

int hold_timeout( int (*f)(void), time_type max_time);
// Tillägg i filen clock.c
int hold_timeout(int (*f)(void), time_type max_time) {
   int c;
   time_type stop;

// vänta tills f() är sann eller högst max_time ms
// om max_time < 0 finns ingen timeout

   stop = tick + max_time / 10;
   if (max_time > 0)
      while(!(c=f()) && tick < stop)
```

```
        ;
    else
        while( !(c=f() ))
        ;
    return c;
}
```

Uppgift 5b:

```
// I filen alarm_ports.h
#define SENSORREG_ADR      0xA024
#define SENSORREG          *((portptr) SENSORREG_ADR)
#define SENSOR_VEC_ADR     0xFE20   // Adress till avbrottsvektor
#define SENSOR_VEC         *((vecptr) SENSOR_VEC_ADR)
#define sensor_on_bit      0x80
#define sensor_signal_bit  0x40
#define sensor_intr_bit    0x02

// I filen alarm.c
#include "clock.h"
#include "alarm_ports.h"
#include "alarminter.h"   // innehåller deklaration av alarmtrap

void raise_alarm();
static volatile int signal;

void init_alarm() {
  port shadow = 0;
  signal = 0;
  SENSOR_VEC = alarmtrap;
  set(shadow, sensor_on_bit);
  set(shadow, sensor_intr_bit);
  SENSORREG = shadow;
}

void alarm_inter(void) {     // Anropas vid avbrott
  clear(SENSORREG, sensor_signal_bit);
  signal = 1;
}

int signal_on(void) {
  return signal;
}

int main() {
  init_clock();
  init_alarm();
  while (1) {
    if (signal) {   // första signal
      signal = 0;
      // vänta på andra signal
      if (hold_timeout(signal_on, 10000)) {
        signal = 0;
        raise_alarm();
      }
    }
  }
}
```